Leaky regions

Tim Harris

Microsoft Research

Introduction

Hypothesis: even when storage is managed by a garbage collector, it's usually possible to say when a given object can be destroyed

- GC means that:
  - The programmer doesn't have to figure this out
  - It can vary between apps, e.g. for library code
- Plus there are the oft-claimed GC advantages e.g.
  - Allocation is usually fast
  - Object retention leads to bloat rather than crashes

Microsoft Research

Introduction (ii)

- People are still worried about the cost and timing of GC
- From a Singularity discussion:

```
loop {
  BeginTask(resource estimates, deadline);
  ... do time-sensitive work ...
  EndTask(out actual resources used, out slack time);
  waitForNextFrame();
}
```

- Can we identify blocks of code like this BeginTask...EndTask block where objects allocated in the block are expected to be destroyed by the time the block ends?
- Can we exploit this to (i) reduce the cost of GC and (ii) control when work occurs?

Microsoft Research

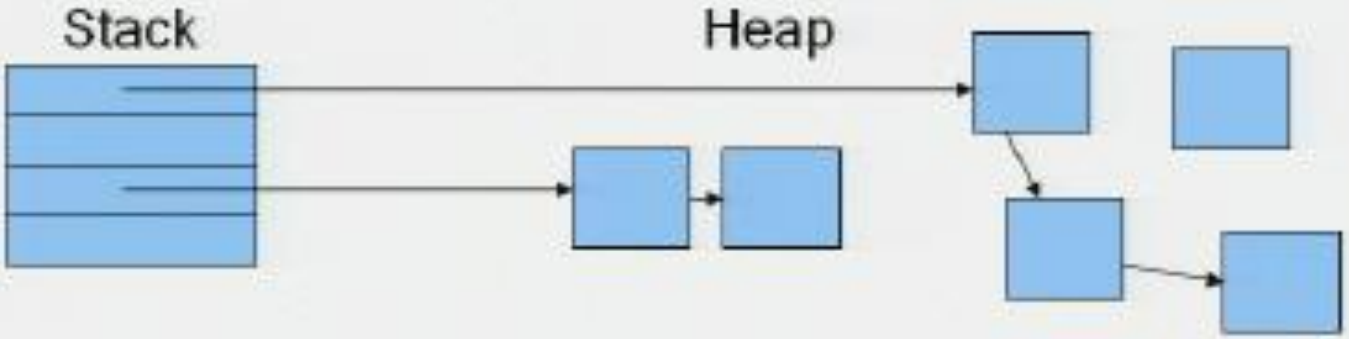
Leaky regions

- Basic idea: annotate method calls saying "most objects allocated in this call will be dead when it returns"
- Safe: runtime checks for violations

Microsoft Research

Leaky regions

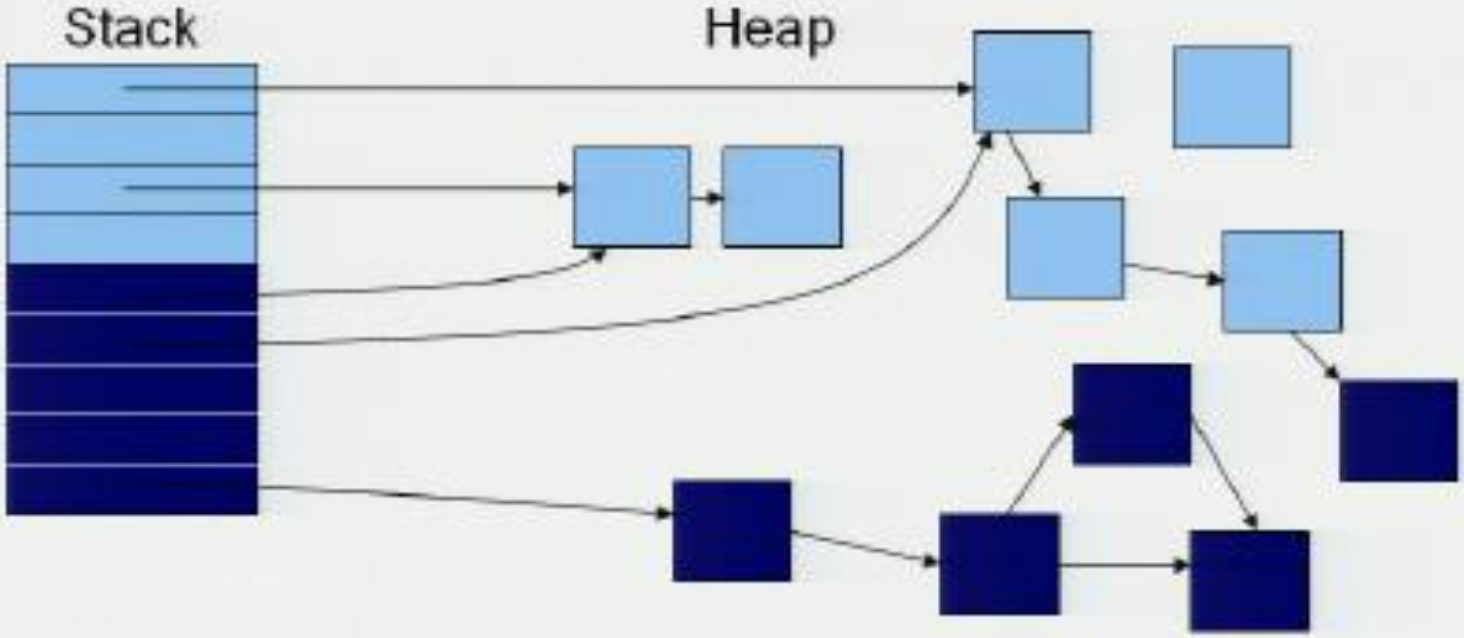
- Basic idea: annotate method calls saying "most objects allocated in this call will be dead when it returns"
- Safe: runtime checks for violations



Microsoft Research

Leaky regions

- Basic idea: annotate method calls saying "most objects allocated in this call will be dead when it returns"
- Safe: runtime checks for violations



Microsoft Research

Leaky regions

- Basic idea: annotate method calls saying "most objects allocated in this call will be dead when it returns"
- Safe: runtime checks for violations
- Three main costs in a naïve system:
  - Checking when an object leaks
  - GC-scope entry/exit code
  - Fragmentation between GC-scopes (e.g. one per page)
- Key assumption:
  - Objects rarely leak from GC-scopes

Microsoft Research

Leaky regions

- Basic idea: annotate method calls saying "most objects allocated in this call will be dead when it returns"
- Safe: runtime checks for violations
- Three main costs in a naïve system:
  - Checking when an object leaks
  - GC-scope entry/exit code
  - Fragmentation between GC-scopes (e.g. one per page)
- Key assumption:
  - Objects rarely leak from GC-scopes

Microsoft Research

Research context

- A lot of existing work on region-based collection (Tofte+Talpin '94)
- More restrictive form in RTSJ

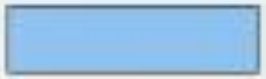
Microsoft Research

Research context

- A lot of existing work on region-based collection (Tofte+Talpin '94)
- More restrictive form in RTSJ

Heap

- Push a new region



Microsoft Research

Research context

- A lot of existing work on region-based collection (Tofte+Talpin '94)
- More restrictive form in RTSJ

- Push a new region

Microsoft Research

Research context

- A lot of existing work on region-based collection (Tofte+Talpin '94)
- More restrictive form in RTSJ

- Push a new region
- Allocate space in any of the regions

Microsoft Research

Research context

- A lot of existing work on region-based collection (Tofte+Talpin '94)
- More restrictive form in RTSJ

- Push a new region
- Allocate space in any of the regions
- Pop & deallocate a region

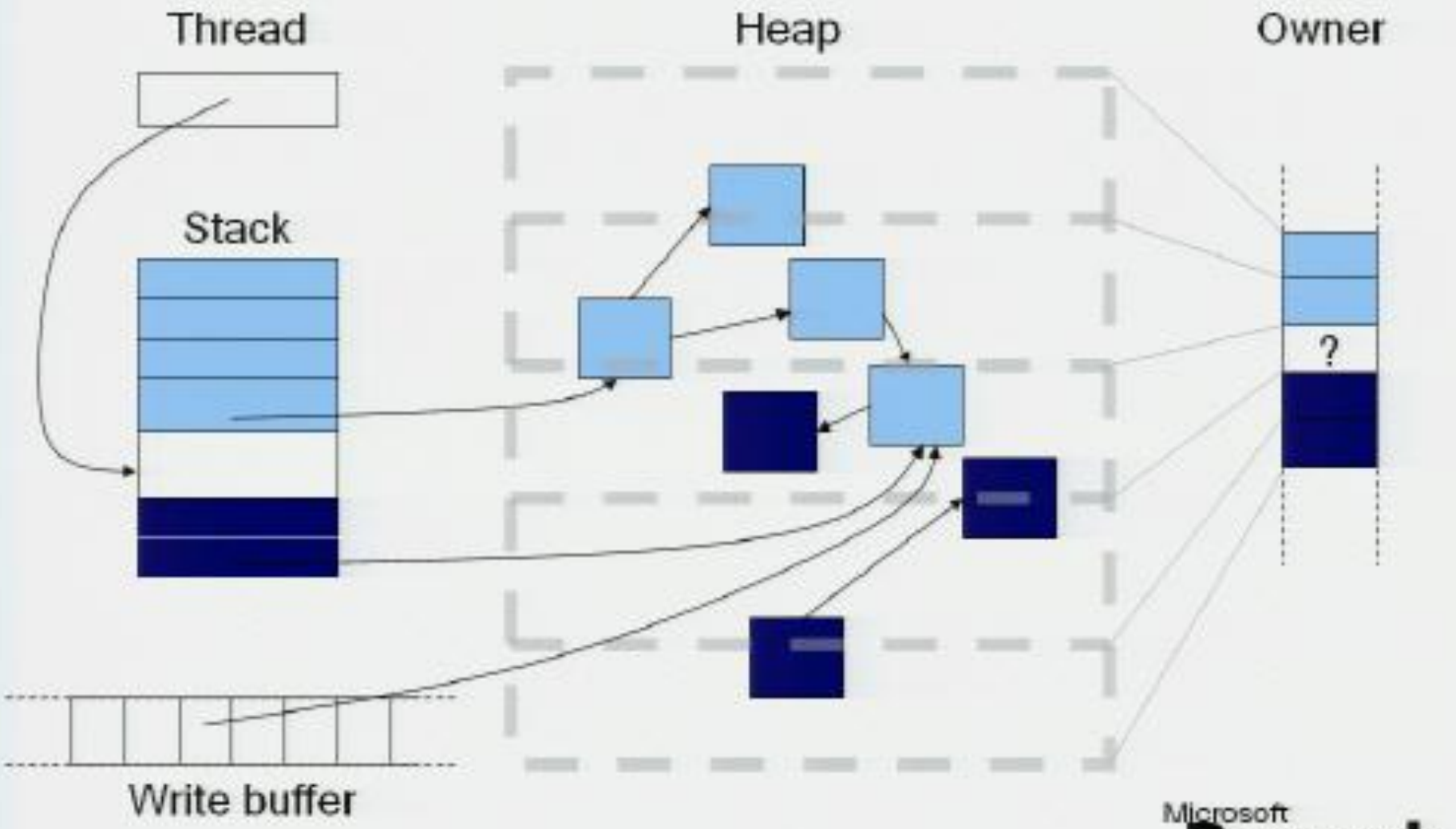
Microsoft Research

Research context (ii)

- Escape analysis for on-stack allocation (OOPSLA '99)
  - As with region systems, must be proven safe
  - Per-method granularity
  - Risks of stack overflow
- Adaptive region-based allocation (Qian+Hendren, ISMM '02)
  - Mark allocation sites escaping / non-escaping
  - Won't tolerate objects that escape and are recaptured
  - Slightly better than per-method granularity
  - Looks at memory-related stats, not performance

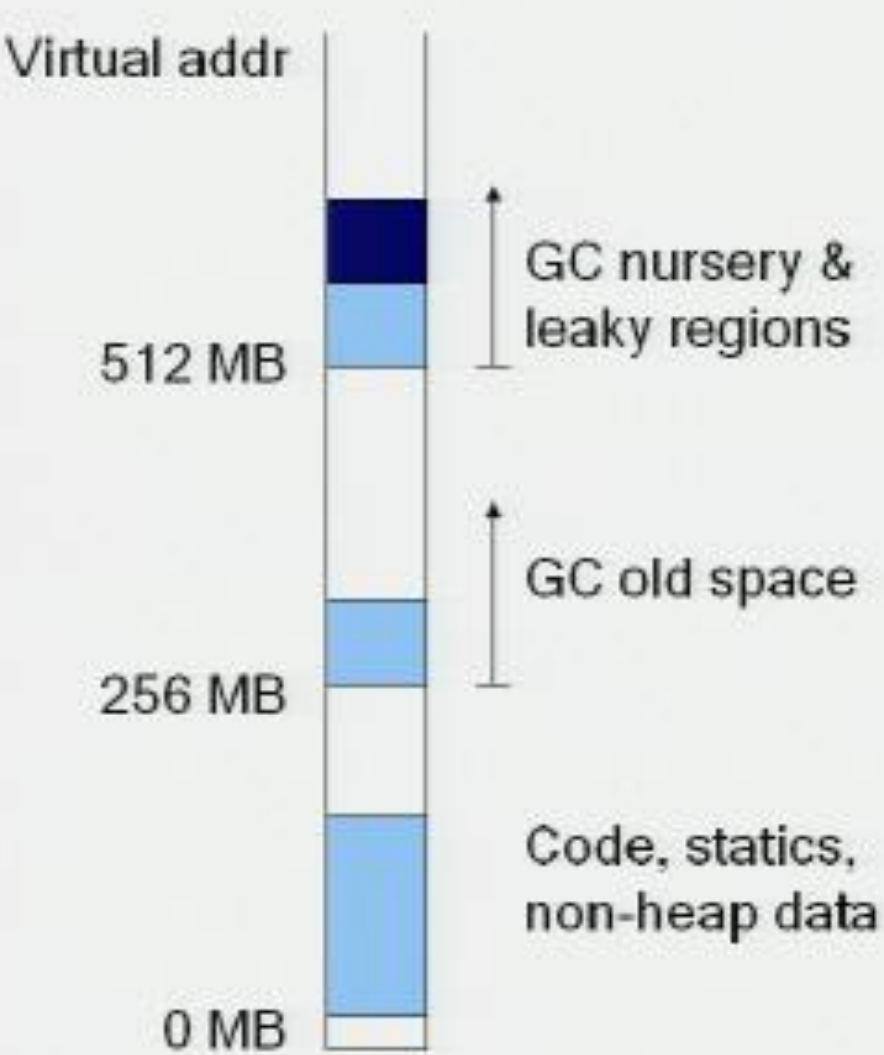
Microsoft Research

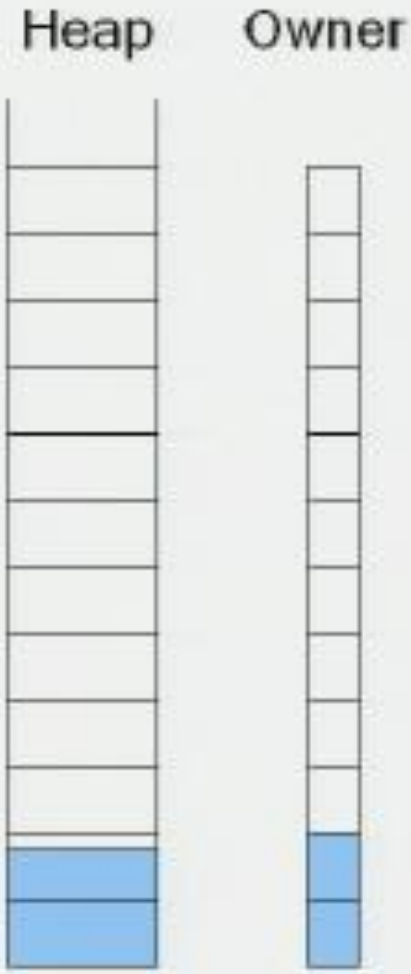
Implementation



Microsoft Research

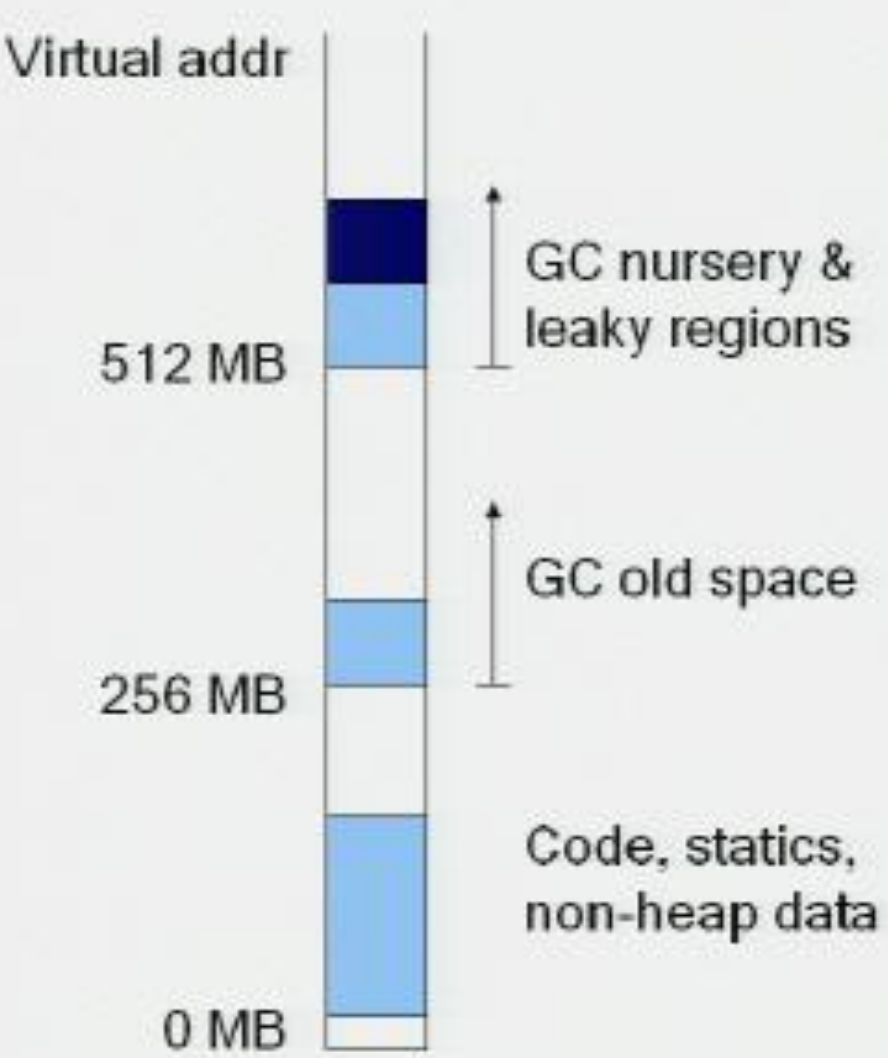
Implementation (ii)

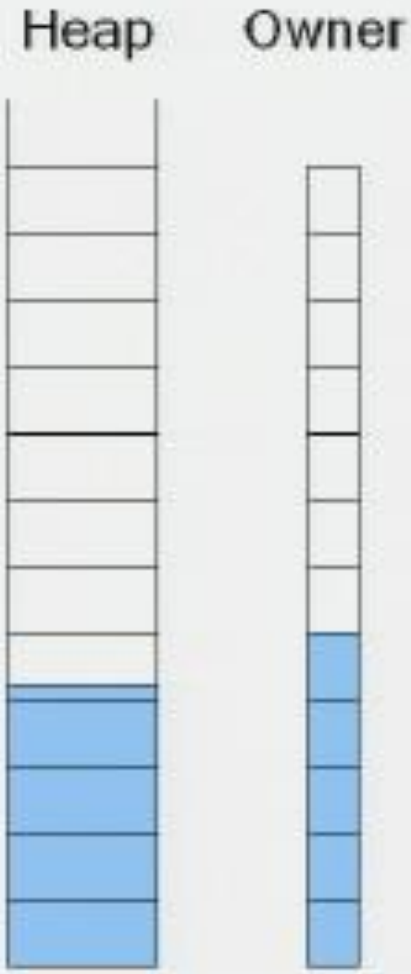




Microsoft Research

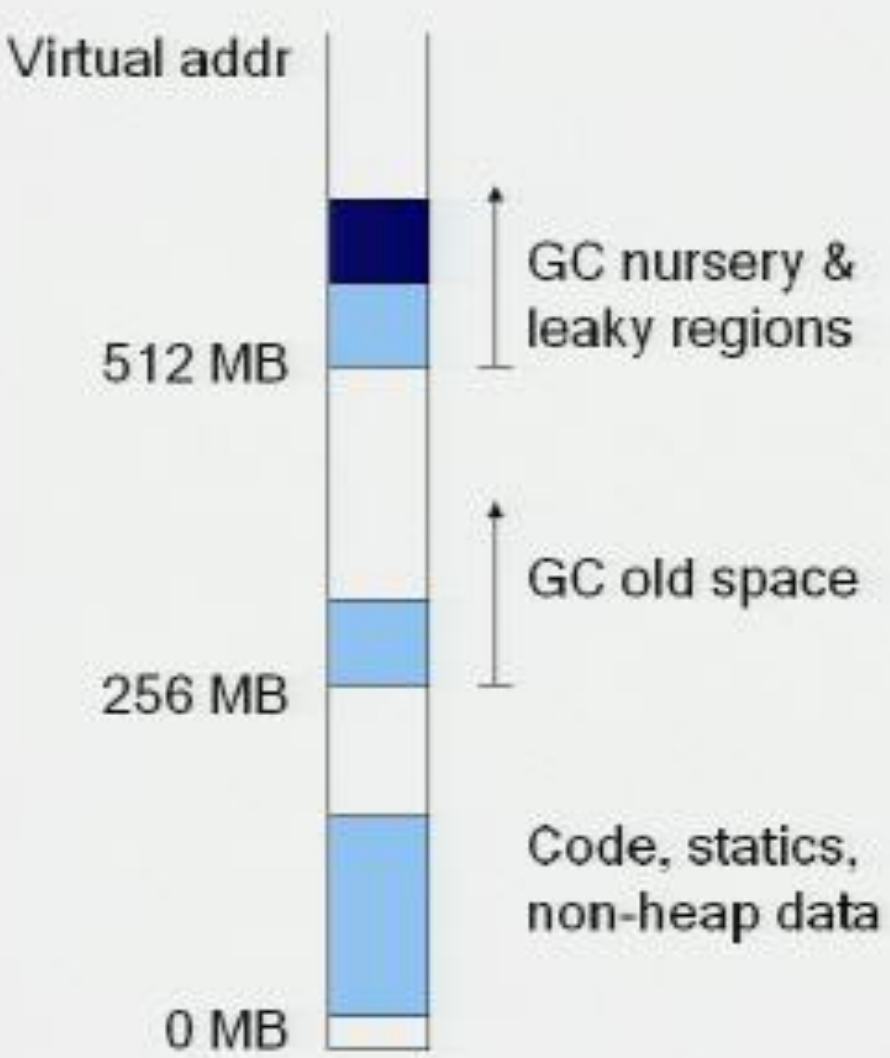
Implementation (ii)

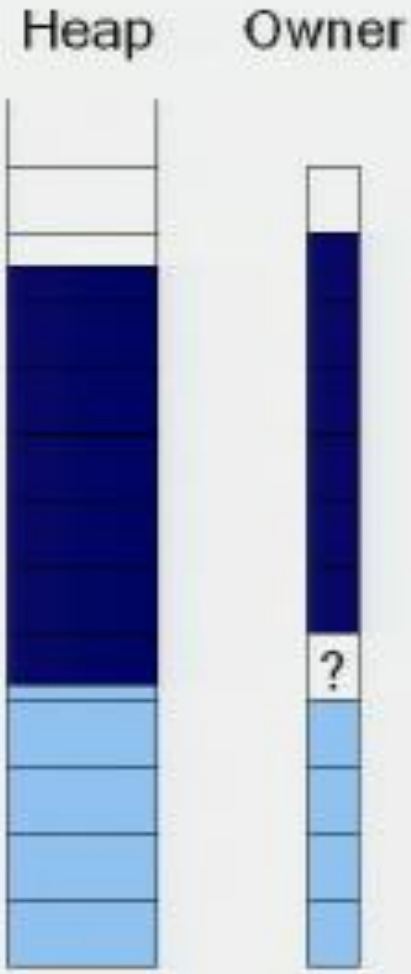




Microsoft Research

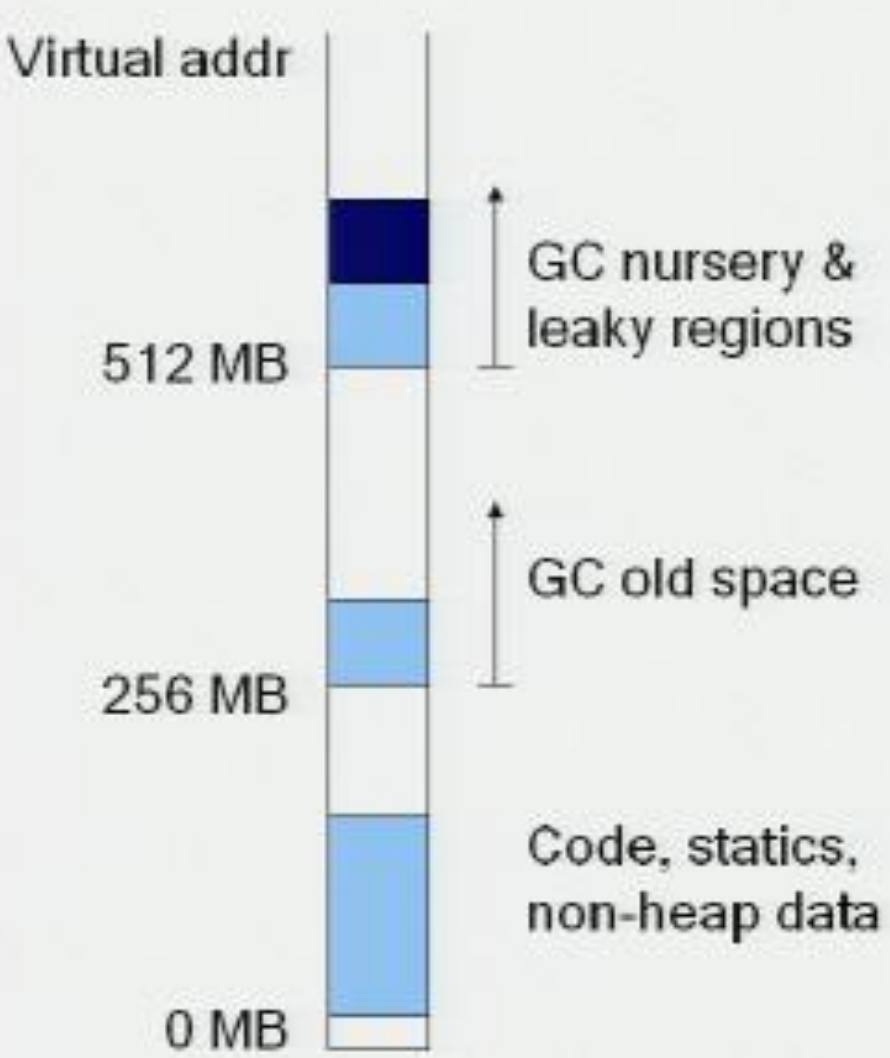
Implementation (ii)

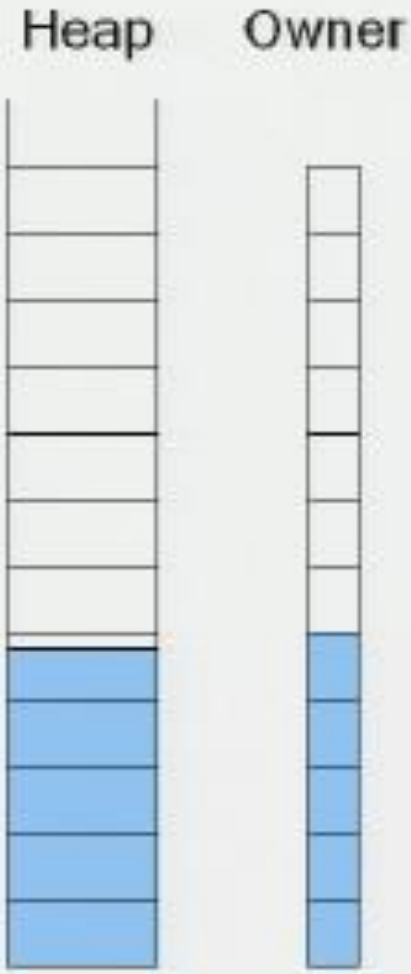




Microsoft Research

Implementation (ii)





Microsoft Research

Annotations (xlisp)

	Calls	Allocation footprint	
impact.li_130.xlisp..ctor -> impact.li_130.xlisp.xlload	20	2061013320	99%
impact.li_130.xlisp.xlload -> impact.li_130.xlisp.xleval	281	2058795532	99%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.evform	41048456	2058795532	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.func	28756528	2058782718	99%
impact.li_130.xlisp.xleverg -> impact.li_130.xlisp.xleval	32000688	2058710020	99%
impact.li_130.xlisp.evfun -> impact.li_130.xlisp.xleverg	14449211	2058704812	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.evfun	2445802	2058703180	99%
impact.li_130.xlisp.xlevlist -> impact.li_130.xlisp.xleval	34219881	2055798568	99%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	2703585	1994337804	96%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	17948325	1521302440	73%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	2445802	1510122408	73%
impact.li_130.xlisp.tagblock -> impact.li_130.xlisp.xleverg	4603864	1339567852	64%
impact.li_130.w_xdo.func -> impact.li_130.xlisp.doloop	112743	1335780852	64%
impact.li_130.xlisp.doloop -> impact.li_130.xlisp.tagblock	2562724	1334083528	64%
impact.li_130.xlisp.xlgetvalue -> impact.li_130.xlisp.xlxgetvalue	78751788	1280034392	61%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.xlgetvalue	78731784	1259714328	61%
impact.li_130.w_xlet.func -> impact.li_130.xlisp.let	90581	1205580718	58%
impact.li_130.xlisp.let -> impact.li_130.xlisp.xleverg	159334	1204765380	58%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	5094789	1006555560	48%
impact.li_130.w_xor.func -> impact.li_130.xlisp.xleverg	54783	770584288	37%
impact.li_130.w_xprogn.func -> impact.li_130.xlisp.xleverg	382864	710183472	34%

Annotations (xlisp)

	Calls	Allocation footprint	
impact.li_130.xlisp..ctor -> impact.li_130.xlisp.xlload	20	2061013320	99%
impact.li_130.xlisp.xlload -> impact.li_130.xlisp.xleval	281	2058795532	99%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.evform	41048456	2058795532	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.func	28756526	2058782718	99%
impact.li_130.xlisp.xleverg -> impact.li_130.xlisp.xleval	32000688	2058710020	99%
impact.li_130.xlisp.evfun -> impact.li_130.xlisp.xleverg	14449211	2058704812	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.evfun	2445802	2058703180	99%
impact.li_130.xlisp.xlevlist -> impact.li_130.xlisp.xleval	34219881	2055798568	99%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	2703585	1994337804	96%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	17948325	1521302440	73%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	2445802	1510122408	73%
impact.li_130.xlisp.tagblock -> impact.li_130.xlisp.xleverg	4603864	1339567852	64%
impact.li_130.w_xdo.func -> impact.li_130.xlisp.doloop	112743	1335780852	64%
impact.li_130.xlisp.doloop -> impact.li_130.xlisp.tagblock	2562724	1334083528	64%
impact.li_130.xlisp.xlgetvalue -> impact.li_130.xlisp.xlxgetvalue	78751788	1280034392	61%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.xlgetvalue	78731784	1259714328	61%
impact.li_130.w_xlet.func -> impact.li_130.xlisp.let	90581	1205580718	58%
impact.li_130.xlisp.let -> impact.li_130.xlisp.xleverg	159334	1204765380	58%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	5094799	1006555560	48%
impact.li_130.w_xor.func -> impact.li_130.xlisp.xleverg	54793	770584288	37%
impact.li_130.w_xprogn.func -> impact.li_130.xlisp.xleverg	382864	710183472	34%

Annotations (xlisp)

	Calls	Allocation footprint	
impact.li_130.xlisp..ctor -> impact.li_130.xlisp.xiload	20	2061013320	99%
impact.li_130.xlisp.xiload -> impact.li_130.xlisp.xleval	281	2058795532	99%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.evform	41048456	2058795532	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.[illegible]	[illegible]	[illegible]	[illegible]
impact.li_130.xlisp.xleverg -> impact.li_130.xlisp.xleval	32000668	2058710020	99%
impact.li_130.xlisp.evfun -> impact.li_130.xlisp.xleverg	14449211	2058704812	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.evfun	2445802	2058703180	99%
impact.li_130.xlisp.xlevlist -> impact.li_130.xlisp.xleval	34219881	2055798568	99%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	2703585	1994337604	96%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	17948325	1521302440	73%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	2445802	1510122408	73%
impact.li_130.xlisp.tagblock -> impact.li_130.xlisp.xleverg	4603984	1339567852	64%
impact.li_130.w_xdo.func -> impact.li_130.xlisp.doloop	112743	1335780852	64%
impact.li_130.xlisp.doloop -> impact.li_130.xlisp.tagblock	2562724	1334083528	64%
impact.li_130.xlisp.xlgetvalue -> impact.li_130.xlisp.xlxgetvalue	78751788	1280034392	61%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.xlgetvalue	78731784	1259714328	61%
impact.li_130.w_xlet.func -> impact.li_130.xlisp.let	90581	1205580716	58%
impact.li_130.xlisp.let -> impact.li_130.xlisp.xleverg	159334	1204765380	58%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	5094789	1006555560	48%
impact.li_130.w_xor.func -> impact.li_130.xlisp.xleverg	54793	770584288	37%
impact.li_130.w_xprogn.func -> impact.li_130.xlisp.xleverg	382864	710183472	34%

Annotations (xlisp)

	Calls	Allocation footprint	
impact.li_130.xlisp._ctor -> impact.li_130.xlisp.xlload	20	2061013320	99%
impact.li_130.xlisp.xlload -> impact.li_130.xlisp.xleval	281	2058795532	99%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.evform	41048456	2058795532	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.func	28756528	2058782718	99%
impact.li_130.xlisp.xleverg -> impact.li_130.xlisp.xleval	32000688	2058710020	99%
impact.li_130.xlisp.evfun -> impact.li_130.xlisp.xleverg	14449211	2058704812	99%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.evfun	2445902	2058703180	99%
impact.li_130.xlisp.xlevlist -> impact.li_130.xlisp.xleval	34219891	2055798568	99%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	2703585	1994337604	96%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	17948325	1521302440	73%
impact.li_130.xlisp.evform -> impact.li_130.xlisp.xlevlist	2445902	1510122408	73%
impact.li_130.xlisp.tagblock -> impact.li_130.xlisp.xleverg	4603984	1339567852	64%
impact.li_130.w_xdo.func -> impact.li_130.xlisp.doloop	112743	1335780652	64%
impact.li_130.xlisp.doloop -> impact.li_130.xlisp.tagblock	2562724	1334083528	64%
impact.li_130.xlisp.xlgetvalue -> impact.li_130.xlisp.xlxgetvalue	78751788	1280034392	61%
impact.li_130.xlisp.xleval -> impact.li_130.xlisp.xlgetvalue	78731784	1259714328	61%
impact.li_130.w_xlet.func -> impact.li_130.xlisp.let	90581	1205590716	58%
impact.li_130.xlisp.let -> impact.li_130.xlisp.xleverg	159334	1204765380	58%
impact.li_130.w_xcond.func -> impact.li_130.xlisp.xleverg	5094789	1006555560	48%
impact.li_130.w_xor.func -> impact.li_130.xlisp.xleverg	54783	770584288	37%
impact.li_130.w_xprogn.func -> impact.li_130.xlisp.xleverg	382864	710183472	34%

Annotations (xlisp)

- impact.li_130.w_xdo.func -> impact.li_130.xlisp.doloop:
  - 112743 invocations
  - 1215MB allocated
  - 1210MB reclaimed
  - 0.4MB leaked
- impact.li_130.xlisp.evform -> impact.li_130.xlisp.evfun
  - 2445902 invocations
  - 1209MB allocated
  - 807MB reclaimed
  - 0.3MB leaked
- Many objects leaked temporarily through static fields
- Leaked objects are lisp heap segments

Microsoft Research

Annotations (crafty)

- Crafty.Net.OO.MoveGenerator.Search -> Crafty.Net.OO.MoveGenerator.ABSearch
  - 84767 invocations
  - 217MB allocated
  - 217MB reclaimed
  - 0.5MB leaked
- 87% allocation footprint
- Leaks come from checking for board-state repetitions

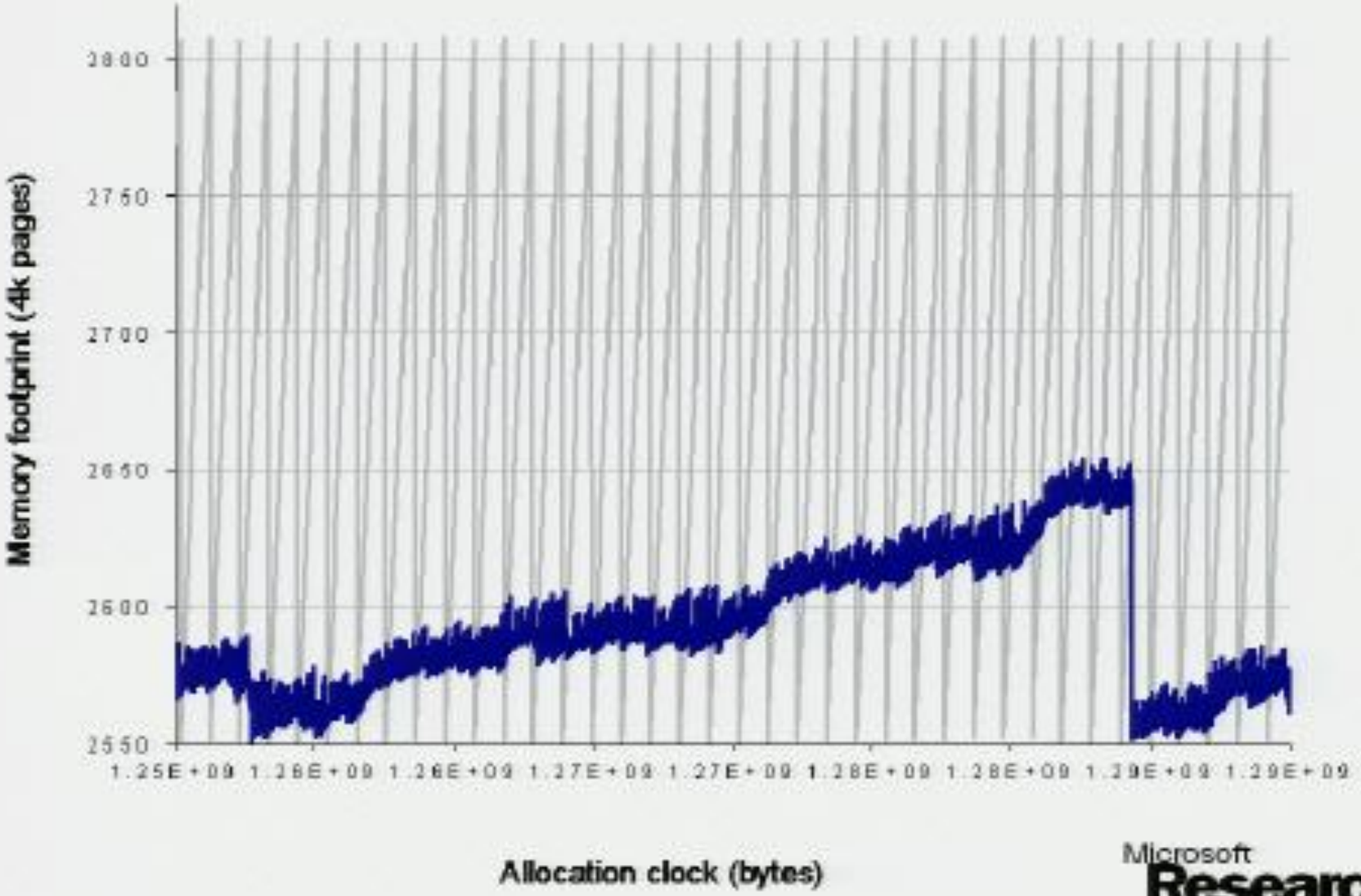
Microsoft Research

Annotations (go, ijpeg)

- impact.go.go.Main -> impact.go.go.getmove
  - 296 invocations
  - 709MB allocation footprint
  - 704MB reclaimed
  - No leaks
  - Fairly large allocation volumes per invocation => little reclamation with small young generation
- Two calls to go_execute_compression_and_decompression
  - 128 invocations
  - 28MB allocation footprint
  - 28MB reclaimed
  - 0.07MB leaks (arrays in spec_define_subimage_int)

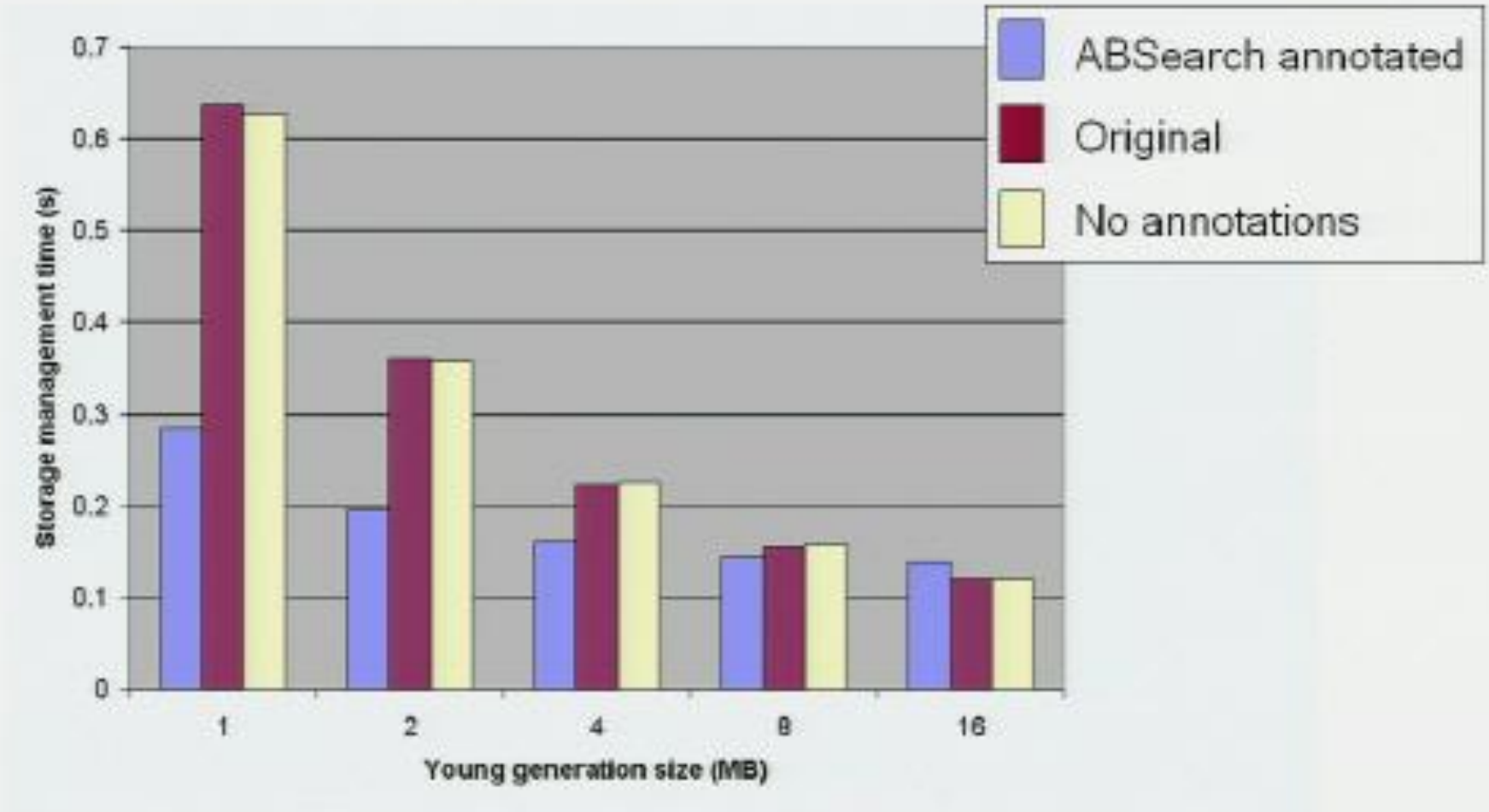
Microsoft Research

Results (xlisp)



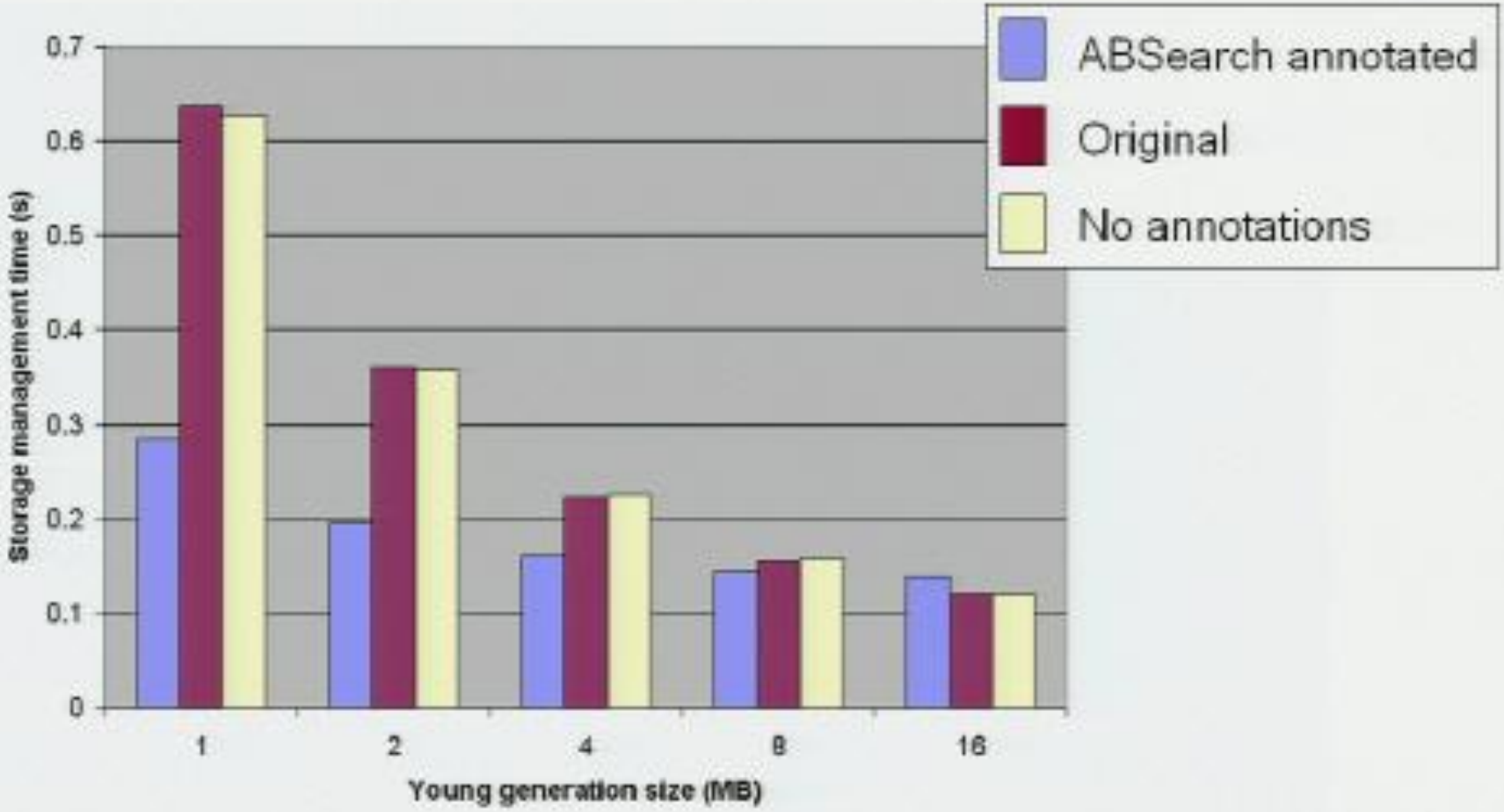
Microsoft Research

Results (crafty)



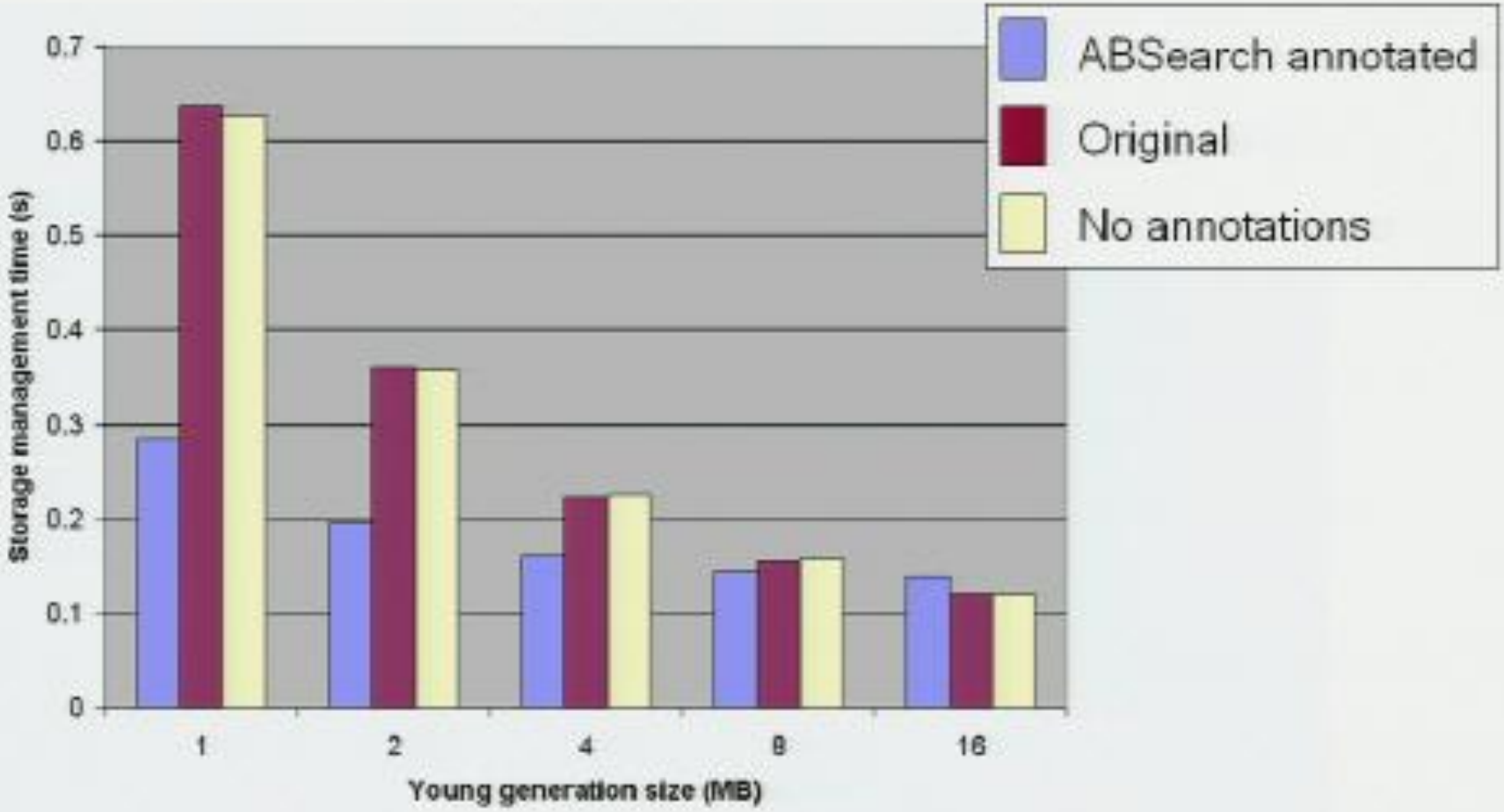
Microsoft Research

Results (crafty)



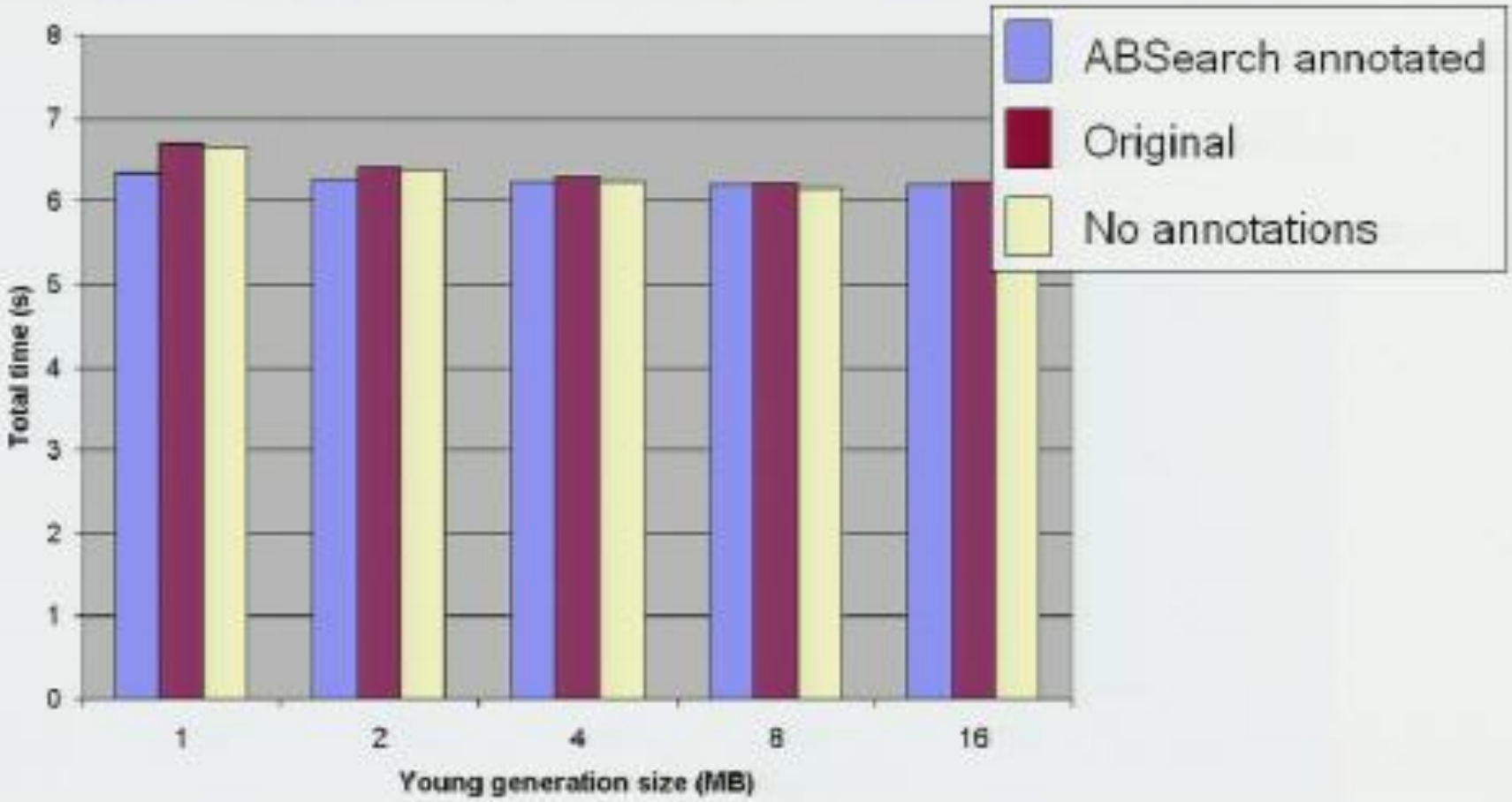
Microsoft Research

Results (crafty)



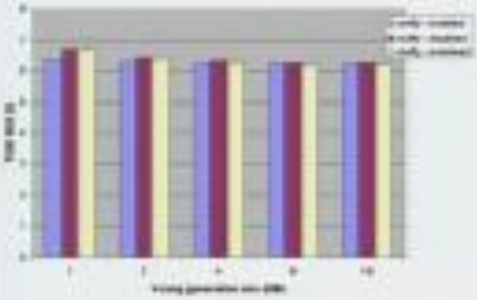
Microsoft Research

Results (crafty)

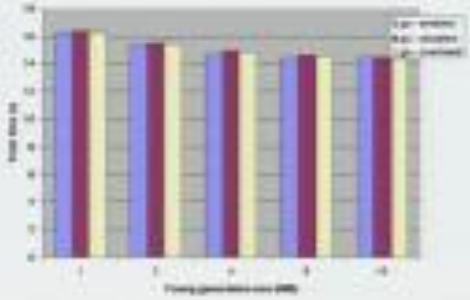


Microsoft Research

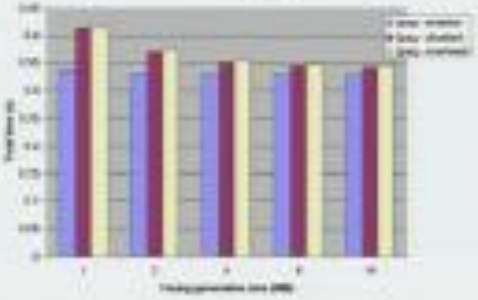
crafty



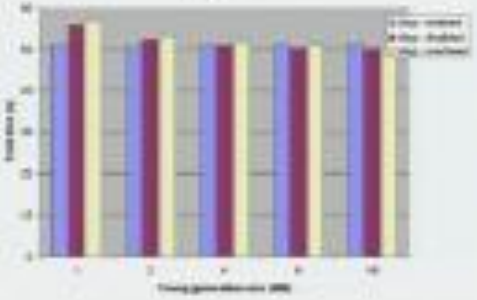
go

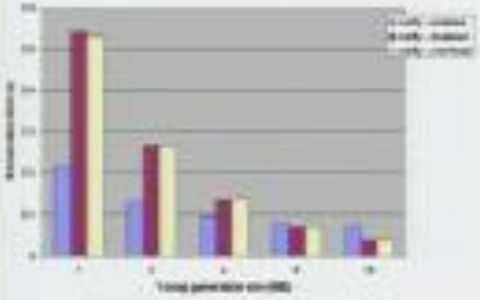


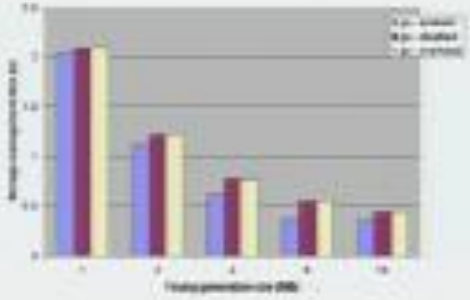
ijpeg

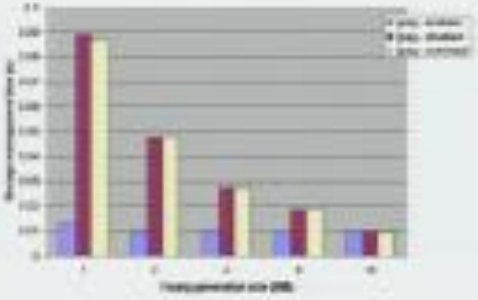


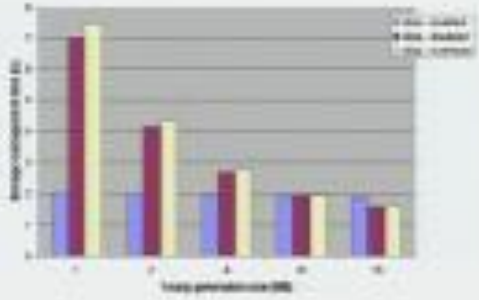
xlisp

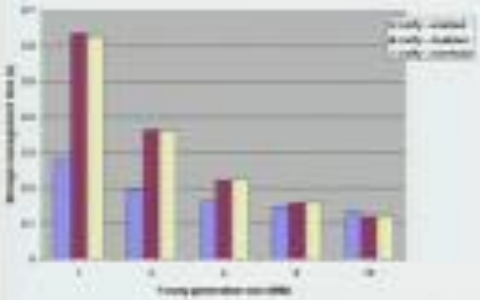


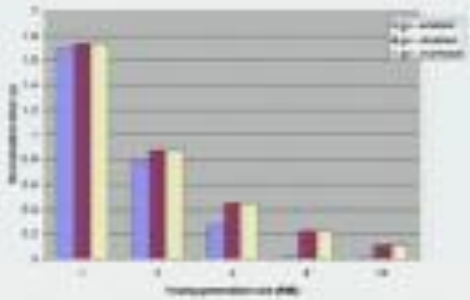


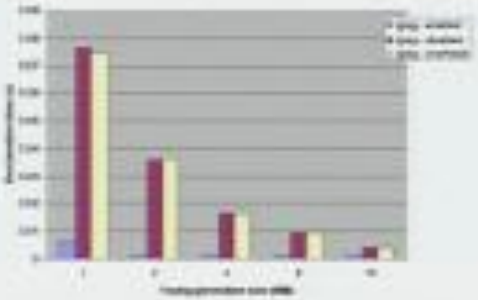


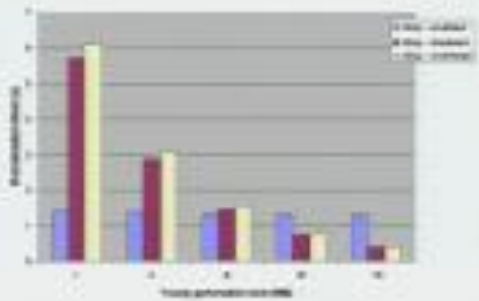












Microsoft Research

Conclusions

- Practical run-time costs unlike previous work
- Evaluation shows the value of
  - Recapturing escapees
  - Considering more than individual method calls
- Picking the boundary is important
  - Few calls, little leakage, large portion of total allocation
  - Scope for feedback-directed choice
- Evaluation with other collectors
  - Other kinds of write barrier
- This is all single threaded...

Microsoft Research